AU 332 Artificial Intelligence: Principles and Techniques

By: Guowei Deng

Instructor: Yue Gao

October 20, 2018

I. INTRODUCTION

The homework of this week is much harder than the last week. We are to solve 3 problems and implement 3 algorithms using DFS, BFS, UCS and A* algorithm.

It's quite important for us to well understand these algorithms.

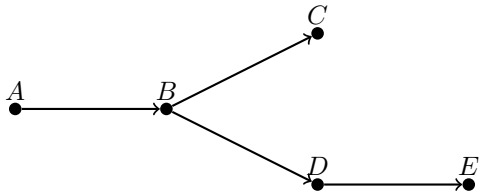
- DFS(Depth First Search) always expands the deepest node in the current frontier of the search tree using FIFO queue. The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors. As those nodes are expanded, they are dropped from the frontier, so then the search backs up to the next deepest node that still has unexplored successors.
- BFS(Breadth-first search) is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on. The algorithm is achieved very simplyby using a FIFO queue for the frontier.
- UCS(uniform-cost search) expands the node n with the lowest path cost g(n). This is done by storing the frontier as a priority queue ordered by g.
- A* algorithm can also be done by storing the friontier as a priority queue.However, A* evaluates nodes by combining g(n), the cost to reach the node, and h(n), the cost to get from the node to the goal:

$$f(n) = g(n) + h(n)$$

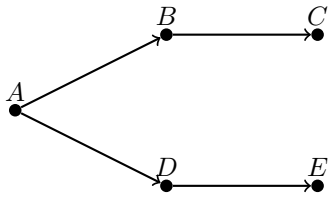
II. SOLUTION

A. Graph Traversal

- DFS:



- BFS:



- complexity: The time complexity of BFS and DFS for a same graph is also the same. Because in the worst condition, both algorithms have to traversa all the nodes and edges in the graph. For this graph, it has n nodes and the maximum degree for each node is d. So the time complexity should equal to O(nd).
  In the worst condition for DFS alogorithm, it has to store n nodes and $\frac{nd}{2}$, so the space complexity is $O(nd)$. And so do BFS.

B. Uniform Cost Search Algorithm

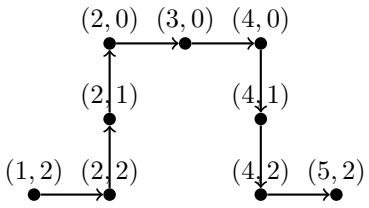
- fringe list:
  $\emptyset \to A, 0$
  $A \to B, 10 \quad A \to C, 3 \quad A \to D, 20$
  $A \to B, 10 \quad C \to B, 5 \quad C \to E, 18 \quad A \to D, 20$
  $A \to B, 10 \quad B \to D, 10 \quad C \to E, 18 \quad A \to D, 20$
  $B \to D, 10 \quad C \to E, 18 \quad A \to D, 20 \quad B \to D, 15$
  $D \to E, 21 \quad C \to E, 18 \quad A \to D, 20 \quad B \to D, 15$
  $D \to E, 21 \quad C \to E, 18 \quad A \to D, 20 \quad D \to E, 26$
  $D \to E, 21 \quad A \to D, 20 \quad D \to E, 26$

- closed list:
  $\emptyset \to A, 0$
  $A \to B, 10 \quad A \to C, 3 \quad A \to D, 20$
  $C \to B, 5 \quad C \to E, 18 \quad A \to D, 20$
  $B \to D, 10 \quad C \to E, 18 \quad A \to D, 20$
  $D \to E, 21 \quad C \to E, 18 \quad A \to D, 20$
  $D \to E, 21 \quad A \to D, 20$

C. A* Algorithm

Heuristic function: $h(n) = \sqrt{(X_{goal} - X_{now})^2 + (Y_{goal} - Y_{now})^2}$

- path:



D. Programming Assignment

1. *BFSvsDFS*

```
Large graph
came from DFS {'A': 'S', 'B': 'A', 'D': 'B', 'F': 'D', 'H': 'F', 'G': 'H', 'E': 'G', 'S': None}
path from DFS ['S', 'A', 'B', 'D', 'F', 'H', 'G', 'E']
came from BFS {'S': None, 'A': 'S', 'C': 'S', 'B': 'A', 'D': 'A', 'L': 'C', 'H': 'B', 'F': 'D', 'I': 'L', 'J': 'L',
 'G': 'H', 'K': 'I', 'E': 'G'}
path from BFS ['S', 'A', 'B', 'H', 'G', 'E']
Small graph
came from DFS {'B': 'A', 'D': 'B', 'E': 'D', 'A': None}
path from DFS ['A', 'B', 'D', 'E']
came from BFS {'A': None, 'B': 'A', 'D': 'A', 'C': 'B', 'E': 'D'}
path from BFS ['A', 'D', 'E']
```

2. *UniformCostSearch*

```
Small graph
came from UCS  {'A': None, 'B': 'A', 'D': 'A', 'C': 'B', 'E': 'D'}
cost form UCS  {'A': 0, 'B': 2, 'D': 4, 'C': 5, 'E': 7}
path from UCS  ['A', 'D', 'E']
Large graph
came from UCS  {'S': None, 'A': 'B', 'B': 'S', 'C': 'S', 'D': 'B', 'H': 'B', 'L': 'C', 'F': 'H', 'G': 'H', 'E': 'G'
, 'I': 'L', 'J': 'L'}
cost form UCS  {'S': 0, 'A': 5, 'B': 2, 'C': 3, 'D': 6, 'H': 3, 'L': 5, 'F': 6, 'G': 5, 'E': 7, 'I': 9, 'J': 9}
path from UCS  ['S', 'B', 'H', 'G', 'E']
```

3. *AStarSearch*

```
Small Graph
The graph does not satisfy the consistency
came from Astar  {'A': None, 'B': 'A', 'D': 'A', 'E': 'D'}
cost form Astar  {'A': 0, 'B': 2, 'D': 4, 'E': 7}
path from Astar  ['A', 'D', 'E']
Large Graph
The graph does not satisfy the consistency
came from Astar  {'S': None, 'A': 'B', 'B': 'S', 'C': 'S', 'D': 'B', 'H': 'B', 'F': 'H', 'G': 'H', 'E': 'G'}
cost form Astar  {'S': 0, 'A': 5, 'B': 2, 'C': 3, 'D': 6, 'H': 3, 'F': 6, 'G': 5, 'E': 7}
path from Astar  ['S', 'B', 'H', 'G', 'E']
```

E. Extra credit

1. *Valid node*

To check whether the goal and start state are valid nodes in the graph, I think there are two points we need to detect.

1. Detect whether the start node and goal node are in the graph
   The best way to detect this point, we can define some method objects in the class *Graph*. In this way, every time the algorithm want to get imformation such as node's edges and location, we can detect whether the node is in the graph. If not, print "The node is not in the graph"

2. Detect whether there's a way from the start to the goal node.
   If the algorithm has traversaed all the nodes and edges but still doesn't find a path, the node won't in dictionay *came_from*'s keys. So if the *came_from[goal]* goes wrong, print "There's no path from start to goal".

2. *Check consistency*

To check the graph satisfies the consistency of heuristics, we need to define a method object *checkConsistency* in class *Graph*. Once we want to check a graph's consistency, we just use code *Graph.checkConsistency(goal)*

```
def checkConsistency(self, goal):
    for node1 in self.edges:
        for node2 in self.edges[node1]:
            h1 = heuristic(self, node1, goal)
            h2 = heuristic(self, node2, goal)
            cost = self.get_cost(node1, node2)
            if fabs(h1 - h2) > cost:
                print('The graph does not satisfy the consistency')
                return False
    print('The graph satisfies the consistency')
```